Carpet Unrolling for Character Control On Uneven Terrain

Mark Miller[1] Daniel Holden[1] Rami Al-Ashqar[1] Christophe Dubach[1] Kenny Mitchell[2] Taku Komura[1]
[1]University of Edinburgh [2]Edinburgh Napier University

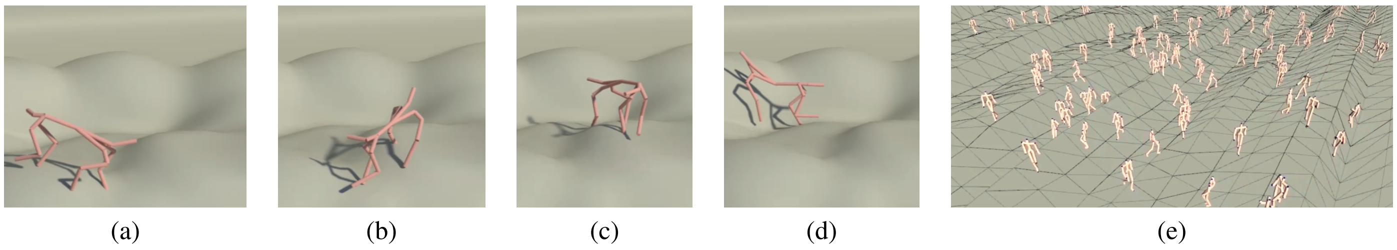


Figure 1: *(a)-(d) Snapshots of a quadruped character walking over an uneven terrain and (e) a snapshot of a large number of biped characters walking on uneven terrain in real-time.*

Abstract

We propose a type of relationship descriptor based on carpet unrolling that computes the joint positions of a character based on the sum of relative vectors originating from a local coordinate system embedded on the surface of a carpet. Given a terrain that a character is to walk over, the carpet is *unrolled* over the surface of the terrain. The carpet adapts to the geometry of the terrain and curves according to the trajectory of the character. Because trajectories of the body parts are computed as a weighted sum of the relative vectors, the character can smoothly adapt to the elevation of the terrain and the horizontal curves of the carpet. The carpet relationship descriptors are easy to parallelize and hundreds of characters can be animated in real-time by making use of the GPUs. This makes it applicable to real-time applications such as computer games.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: locomotion, relationship descriptors, animation, character animation, video games, computer games

1 Introduction

In computer animation and computer games, scenes where characters walk over terrains with arbitrary geometry are very common. Characters need to walk over tilted terrains, stairs, and uneven rocky terrains, while avoiding obstacles, self-collisions and collisions with other characters. Although humans can easily adapt to terrain with different geometry on-the-fly, synthesizing such movements for virtual characters is not straight-forward.

One classical approach to synthesize such movements is to use inverse kinematics. Given a canonical walking cycle over a terrain, the feet trajectories can be adjusted such that the character walks over it. This requires specifying the moments that the feet are in contact with the ground, and producing a trajectory that does not collide with the ground. Also, the movements of the other parts of the body, including the pelvis, torso and the arms must be adjusted to make the motion appear natural.

Another approach is to apply data driven methods. Movements to walk over various types of terrain can be pre-captured and blended to produce movements that satisfies the constraints due to the terrain. This requires capturing various movements and blending them together. Various approaches based on radial basis functions and Gaussian processes have been proposed for such applications. Capturing and managing many types of movements is cumbersome and memory intensive. It is easier if a canonical motion can be adapted to terrains with different shapes.

In this paper, we solve the problem of terrain locomotion by making use of relationship descriptors [Al-Asqhar et al. 2013] which are known to be useful for retargeting motions to characters of different sizes or to interact with objects with different geometry. We propose a simple and effective approach that we call "carpet unrolling" to adapt the locomotion to terrains with arbitrary shapes. We present that the method is applicable to characters with different topology, including bipedal and quadruped characters.

The approach is highly parallelizable, and easily runs on the GPU. As a result, the trajectories and movements of hundreds of characters can be adapted during runtime, which makes it applicable to real-time applications such as computer games, crown animation, and virtual reality applications.

2 Related Work

In this section, we discuss the work related to character control in crowds and motion adaptation methods.

Character Control in Crowd Animation As crowd animation is a very large area, we limit ourselves to methods that involve close character interactions. Very roughly, crowd animation can be divided into global approaches and local agent-based approaches. Global approaches control agents by producing global maps or potential functions [Treuille et al. 2006] and guiding the characters based on such global structures. Despite the fact they produce optimal trajectories to avoid congestion, recently, there is higher interests in local, agent-based approaches, where the control is decentralized to each individual characters.

Various agent-based controllers have been proposed in the area

MIG '15, November 16 – 18, 2015, Paris, France.
© 2015 ACM. ISBN 978-1-4503-3991-9/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2822013.2822031

of computer animation and crowd modeling. Based on Renold's flocking model [Reynolds 1987], Helbing et al. [2000] propose an energy-based method that drives the characters toward their goal while avoiding each other. Simulations of panicking crowd are produced, and such an approach is useful for evaluating the safety of buildings. Reciprocal velocity obstacle [van den Berg et al. 2008] is an approach that is widely adopted in robotics and character animation to control the characters to avoid other characters in the velocity space. Methods based on synthetic vision [Ondrej et al. 2010] can reproduce phenomena such as crowds producing vortices at the intersection and lane-forming when a crowd of people pass through each other.

In previous crowd animation, each character is mostly treated as a particle or a rigid body and the body movements are simply replayed such that the root of the body follows the trajectory computed from the crowd simulation engine. Our proposed method can be useful for producing an animation of crowds moving over a terrain of arbitrary geometry.

Real-time Motion Adaptation to Different Geometry Recently, there is an increasing interest in adapting character movements and postures to interact with other characters, objects and the environment. A classical approach to edit the motion of the character to adapt to different geometry, such as the terrain, is to apply inverse kinematics [Lee and Shin 1999; Shin et al. 2001], or spacetime optimization [Gleicher 1997]. Simple control of the end effectors based on inverse kinematics or spacetime optimization does not always work well, especially when the motion involves close interactions with the geometry, as unexpected collisions may occur between the body parts.

In order to reproduce more natural movements, data driven approaches are also introduced to follow positional constraints provided by the user. Rose et al. [1998] interpolate movements using radial basis functions (RBF). Kovar et al. [2004] enhance this approach; they search movements of the same classes in the database and produce the user desired movements by interpolating the relevant movements by RBF. Mukai and Kuriyama [2005] apply Gaussian processes to interpolate various movements to reach out, hold objects, and step on stairs of different heights. Mukai [2011] further extends the approach for biped locomotion and propose a data structure called *Motion Rings* to adapt the locomotion to different geometry. Although data driven approaches produce excellent results, they require many data samples and the constraints are not precisely satisfied, especially when the training samples are sparse.

Ho et al. [2010] propose a method to adapt existing motion data to different body sizes and environments of different geometry. This approach is further enhanced by Al-Ashqar et al. [2013] to achieve real-time performance. Based on this approach of relationship descriptors, we present a system that allows the characters to adapt interactively to terrains of different geometry. We also present a parallelisation of the approach such that many characters can be animated in real-time.

3 Relationship Descriptors

In this section, we review the relationship descriptor representation proposed in [Al-Asqhar et al. 2013], This representation is especially useful for reproducing animation with the same context even when the geometry of the object is changed. In our representation, the joint positions are computed by the relative translations from a static set of points called descriptor points. The descriptor points are placed on the carpet as explained in Section 5.

Now, we explain how to compute the joint positions from the descriptor points. Let us define the position of joint j by $\mathbf{p_j}$, and the descriptor points by $(\mathbf{d}_1, ..., \mathbf{d}_N)$ (see Fig. 2). We also obtain the normal, tangent and binormal vectors from the geometry of the surface, which are defined by $(\mathbf{n}_1, ..., \mathbf{n}_N)$, $(\mathbf{t}_1, ..., \mathbf{t}_N)$, and $(\mathbf{b}_1, ..., \mathbf{b}_N)$, respectively. The tangent vectors are computed by simply picking one of the edges connected to the vertex and projecting it to the tangent plane, and the binormal vectors are computed by the cross product of the normal and tangent vectors. Given a motion, we represent the joint positions $\mathbf{p_j}$ relative to $\mathbf{d_i}$ using these three vectors:

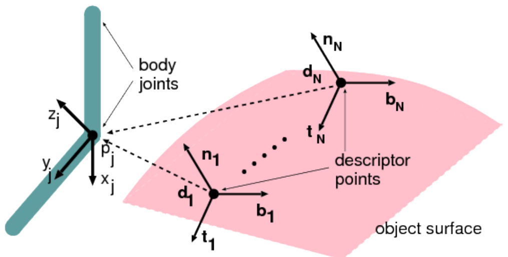


Figure 2: *The local coordinates defined at the body joints and at the sample points on the object.*

$$\mathbf{p_j} = \mathbf{d_i} + \alpha_{i,j}\mathbf{n_i} + \beta_{i,j}\mathbf{t_i} + \gamma_{i,j}\mathbf{b_i}. \quad (1)$$

As we want $\mathbf{p_j}$ to be influenced by not only one but all the descriptor points in proximity, we represent it as the weighted sum of Eq. (1) of all the descriptor points instead:

$$\mathbf{p_j} = \sum_i^N w_{i,j}(\mathbf{d_i} + \alpha_{i,j}\mathbf{n_i} + \beta_{i,j}\mathbf{t_i} + \gamma_{i,j}\mathbf{b_i}) \quad (2)$$

where $w_{i,j}$ is the normalized weight between joint j and descriptor point $\mathbf{d_i}$ whose value is dependent on the distance between the two points and how much the normal vector $\mathbf{n_i}$ is facing toward $\mathbf{p_j}$. For computing the weights, we first calculate the following term for all the descriptor points:

$$w'_{i,j} = \frac{\mathbf{n_i} \cdot (\mathbf{p_j} - \mathbf{d_i})}{\|\mathbf{p_j} - \mathbf{d_i}\|}. \quad (3)$$

The weight fades out as the distance between $\mathbf{p_j}$ and $\mathbf{d_i}$ increases:

$$w''_{i,j} = w'_{i,j} f(\|\mathbf{p_j} - \mathbf{d_i}\|), \quad (4)$$

where

$$f(d) = \begin{cases} 0 \ (r_2^j \le d) \\ 1 - \frac{d - r_1^j}{r_2^j - r_1^j} \ (r_1^j < d < r_2^j) \\ 1 \ (d \le r_1^j), \end{cases} \quad (5)$$

r_1^j is the distance to the closest descriptor point and r_2^j is set to $r_1^j + \frac{1}{4} \times bodyheight$. Finally, we normalize the weights:

$$w_{i,j} = \frac{w''_{i,j}}{\sum_i w''_{i,j}}. \quad (6)$$

Using Eq. (2), the position of each body joint can be reconstructed from the carpet surface. When the surface of the carpet is changed, the updated poses of the body joints can be computed by feeding the mapped descriptor points and the axes of their local coordinates into these equations. More about the reconstruction process is explained in Section 4.

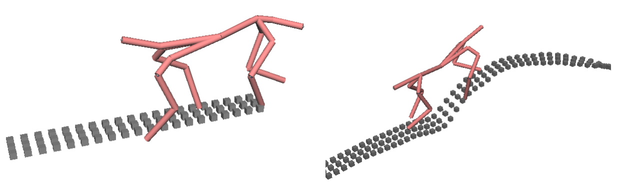


Figure 3: *The descriptors positions before deformation arranged in a grid going out in front of the character (left) and the descriptor positions after deformation bent to follow the trajectory of the character and the surface of the terrain (right).*

4 Motion Adaptation

Using the positions computed by Eq. (2) from the updated position of the descriptor points as the target, we compute the final posture of the character by an iterative inverse kinematics scheme based on [Jakobsen 2001]. This scheme can be roughly broken down into the following four steps: (1) the computation of the affinity, which determines how strongly the joints must be pulled toward the target positions, (2) the force accumulation and integration step, where the effect of external forces such as pushing, pulling etc. are taken into account, and (3) the bone length constraint step, where the joint positions are updated such that the distance between the adjacent joints are kept constant.

Affinity Calculation: The affinity values are computed by summing the weights of the associated descriptors on the surface computed in Eq. (6) and normalizing them:

$$s_j = \frac{\sum_i^N w_{i,j}}{\sum_j^{N_j} \sum_i^N w_{i,j}} \tag{7}$$

where j is index of the joints and N_j is the number of joints.

Force Accumulation and Integration: Instead of explicitly manipulating the joints, we control them by applying virtual forces to the particles that correspond to the joints. The forces are computed by multiplying an elastic constant to the difference between their current and target positions:

$$\mathbf{f}_j = k(\mathbf{p_j^{tar}} - \mathbf{p_j^{cur}}) + \mathbf{f^{ext}}. \tag{8}$$

where k is an elasticity constant that is set to 1, $\mathbf{p_j^{tar}}$ is the target position of the joint computed using the relationship descriptors by Eq. (2), $\mathbf{p_j^{cur}}$ is the current joint position, and $\mathbf{f^{ext}}$ is the external force that is added if an extra effect such as the wind blowing the body needs to be applied.

The target position of the joints are then computed by Verlet integration

$$\mathbf{p_j^{new}} = \mathbf{p_j^{cur}} + d(\mathbf{p_j^{cur}} - \mathbf{p_j^{prev}}) + \frac{1}{2}\mathbf{f_j}\frac{1}{N_s^2} \tag{9}$$

where $\mathbf{p_j^{prev}}$ is the position of the joint in the previous iteration and d is a coefficient that is added to reduce the wobbling effect whose value is set linear to the joint's affinity value ($d = 0.8$ when $s_j = 0$ and $d = 0.2$ when $s_j = 1$).

Constraints Step: Using the updated particle positions $\mathbf{p_j^{new}}$, we compute the final positions of the joints that satisfy the bone-length constraint by iteratively updating the particle positions until the errors of all the constraints are below a certain threshold. To satisfy the bone-length constraints, the positions of each particle is updated by the following equation:

$$\Delta\mathbf{p_j} = \frac{s_k}{s_k + s_j}\frac{\mathbf{p_j} - \mathbf{p_k}}{\|\mathbf{p_j} - \mathbf{p_k}\|}(l^0 - \|\mathbf{p_j} - \mathbf{p_k}\|) \tag{10}$$

where $\mathbf{p_k}$ is a particle that is connected to joint j by a bone, and l^0 is the length of the bone. This will result in joints with large affinity to move less and small affinity to move more.

5 Carpet Unrolling

We now describe about the carpet data structure that we use to adapt the character movements to curves and along the geometry of the terrain. The carpet is represented a regular grid of relationship descriptors that extends outward in front of the character (see Fig. 3). The carpet can be bent or twisted to make the character turn, or projected vertically onto terrain to allow for the character to walk on some surface.

We therefore generate a regular grid of descriptors in front of the character local to character's world space. Given a grid of n by m, grid length l, and grid width w, the position of a descriptor $\mathbf{c}_{ij}$ on the surface of the carpet is then given by the following equation.

$$\mathbf{c}_{ij} = \left(\frac{li}{n}, 0, w\frac{j - \frac{m}{2}}{m}\right) \tag{11}$$

Using the descriptor points local to the character space $\mathbf{c}$ we can find the global descriptor positions $\mathbf{d}$ by multiplying by the character's world matrix $\mathbf{d} = \mathbf{Wc}$. The opposite operation is also possible by using the inverse of the character world matrix. The descriptor positions in local space are given by $\mathbf{c} = \mathbf{W}^{-1}\mathbf{d}$.

Turning It is possible to turn the character by bending the carpet along some curve. As the character always follows the location of the carpet the turning motion will adapt naturally.

We use a NURBS curve to represent the central line of the locomotion. Here we assume the central line of the carpet is along the z axis and the ground is represented by the xz plane. The projection of the descriptor point $\mathbf{c}$ on the z axis can be obtained by $\mathbf{z_c} = \mathbf{c}\dot(0, 0, 1)^T$, and the offset of the descriptor point from the central line can be obtained by $\mathbf{v_c} = \mathbf{c} - \mathbf{z_c}$ (see Fig. 4, left). When the central line is deformed by moving the NURBS curve control points, the descriptor points are recomputed by adding the rotated offset vector to the corresponding projection point (see Fig. 4, right): $\mathbf{c}' = \mathbf{z_c'} + \mathbf{Rv_c}$, where $\mathbf{z_c'}$ is the updated position of the projection point after the curve is deformed, $\mathbf{R}$ is the rotation matrix computed using the tangent direction of the curve at $\mathbf{p_c'}$.

Terrain Adaption To adapt the animation to some terrain the carpet can be projected onto the surface. To do this the descriptors are converted to world space, projected vertically, and then converted back to local space. Given the projection operation $\mathbf{\Phi}$ this is represented as the following $\mathbf{c}' = \mathbf{W}^{-1}\mathbf{\Phi}(\mathbf{Wc})$. Many applications have accelerated data structures for this task such as height-maps, but in the general case, where the terrain consists of triangular polygons, the projection operation can use the barycentric coordinates of each triangle t_1, t_2, t_3 in the xz plane α, β, γ. These are first used to test if a descriptor point $\mathbf{d}$ has xz coordinates which lie within the triangle $0 < \alpha, \beta, \gamma < 1$. If this is the case the barycentric weights can be used to find the projected position in world space $\mathbf{d}' = \alpha t_1 + \beta t_2 + \gamma t_3$. Many triangles can be discarded from this

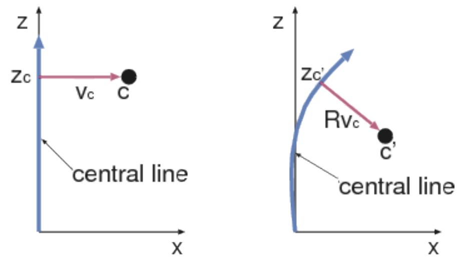
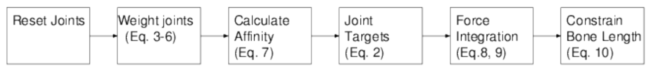


Figure 4: *The descriptor point's offset with respect to the central line is computed in the canonical state (left). When the central line is deformed (right), the descriptor point is computed by adding the rotated offset to the updated projection point on the curve.*



Reset Joints
Weight joints (Eq. 3-6)
Calculate Affinity (Eq. 7)
Joint Targets (Eq. 2)
Force Integration (Eq.8, 9)
Constrain Bone Length (Eq. 10)

Figure 5: *The flowchart of the motion adaptation scheme.*

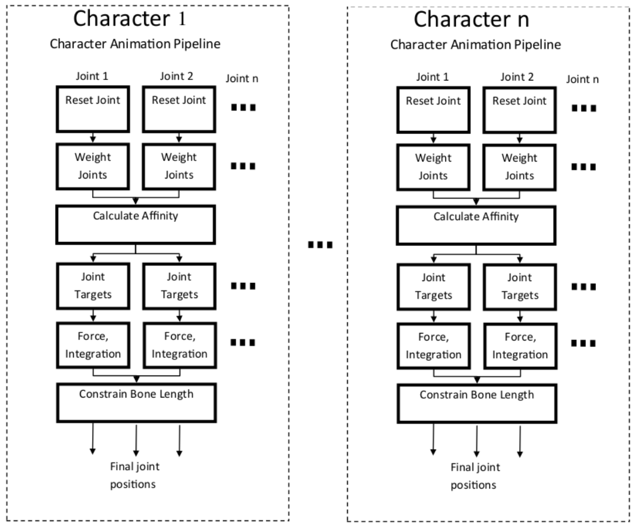


Figure 6: *The parallel pipeline of the system.*

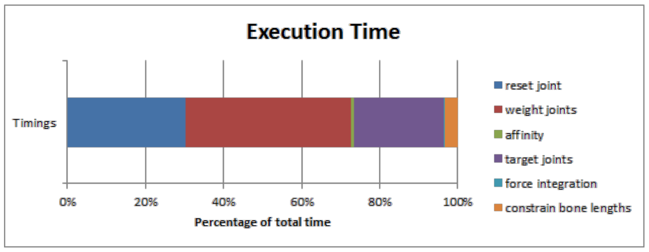


Figure 7: *The breakdown of the execution time.*

test by first calculating their axis aligned bounding boxes and not performing any tests if the descriptor point lies outside of it on the xz plane.

6 Parallel Implementation

Our approach is highly parallelizable and can be accelerated using GPU devices where by many characters are animated. In this section, we briefly describe how we divide the entire task into subtasks that are executed by individual kernels written in the OpenCL language.

For example, computing the target joint locations by summing the relative vectors (Eq. 2) does not require to know the position of other joints. As such, we can evaluate all joints of all characters in the system simultaneously at many stages. As this current implementation of the system does not concern much interaction between characters, we can evaluate the joints of all characters in the system in parallel. As GPUs can execute hundreds of threads concurrently, this allows us to evaluate hundreds of joints very quickly at the same time. This results in extremely large speed ups compared to the original sequential version.

The breakdown of the motion adaptation task is shown in Fig. 5. In all stages, each character is processed independently from the other and in parallel using an OpenCL workgroup. The description of each stage is as follows:

- **Resetting Joints:** This stage is simply concerned with setting the rest position of all the joints for the current frame. It also resets the state of the system. Each joint is processed independently within the workgroup, using one thread per joint.
- **Weighting Joints:** Calculates the weights of the descriptors and their effects on the joint of the system (Eq. 3-6). As above, one thread per joint is used to performed this stage.
- **Calculate Affinity:** Calculate the affinity value (Eq. 7) that determines how stiff the joint should stay in the space. This stage is performed with one thread per character since the joints cannot be processed independently.
- **Joint targets:** This stage deals with computing the target position of the joints based on the deformed descriptor positions and the previously calculated weight and affinity values (Eq. 2). The joints are processed independently using multiple threads.
- **Force integration:** This stage treats each joint as a particle. Forces are applied to the particles by PD control (Eq. 8) and the positions are integrated by Verlet integration (Eq. 10). Each joint is processed independently by a thread.
- **Constrain Bone Lengths:** We constrain the distance between the joints (Eq. 10). This stage needs to be done with one thread per character as the joints need to be traversed iteratively.

The parallel pipeline of the system is shown in Fig. 6, and the breakdown of the execution time is shown in Fig. 7.

7 Results

In this section we present the results of our system by producing animation of characters walking over uneven terrain using the carpet technique. We also present the performance of the system when running it on different parallel setups. The synthesized animation can be viewed in the supplementary video.

Walking Over an Uneven Terrain We use our approach to generate locomotion for a quadruped character. Given a simple walk loop consisting of just 35 frames of animation we generate a natural looking motion of the character walking over terrain and turning in various directions.

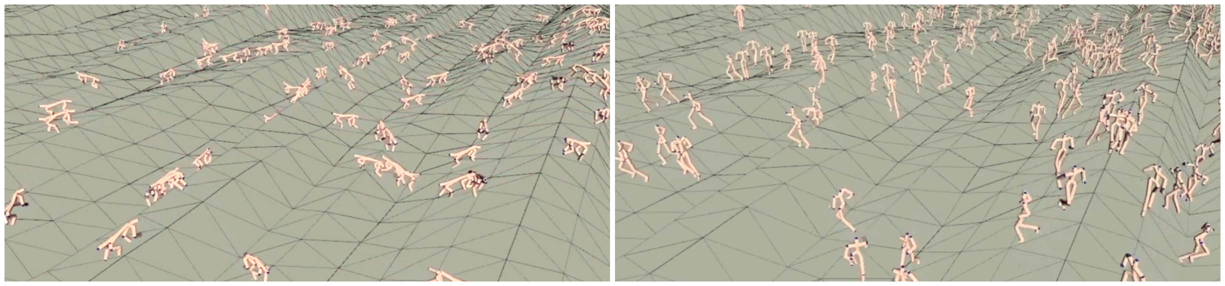


Figure 8: *Using the GPU animation for many characters can be generated. Here we show motion generated in real time for 256 quadrupeds (left) and 256 bipeds (right).*

First the looped walking motion is repeated to generate a long motion of the character walking in a straight line. Relationship descriptors are generated under this motion and the weights for the descriptors are calculated. The descriptors are then deformed along a NURBS curve, which represents the trajectory of the character across the terrain. This introduces the turning into the character's motion. Finally, the descriptors are projected vertically onto the terrain and the joint positions integrated using the descriptor weights. This creates the deformed motion of the character walking along the terrain.

Snapshots of the quadruped walking over an uneven terrain are shown in Fig. 1 (a)-(d). Also as can be viewed in the supplementary video, the quadruped character can adapt well to the uneven terrain and conduct a natural gait cycle.

Evaluation of Parallelism Four primary setups were tested: CPU single thread, CPU multi-thread, Integrated Intel GPU (20 cores). and dedicated Nvidia GPU (2688 cores). By testing on the different devices available in the system, we gain insight into which may be best for this specific operation. For the evaluation of the performance, 100 frames were measured and an average taken. We tested with a number of characters ranging from 1 to 512. The hardware used is composed of i7-4790k 4.0GHz CPU, 16GB RAM and a Geforce Titan with 6GB GRAM. The system is written in OpenCL 1.2 and compiled in Visual Studio 2013 on Windows 8.1.

The performance of the system in different setups are shown in Table 1; it can be observed that parallelism provides a significant speedup. The integrated GPU system shows better performance up to ten characters, as it shares the host memory with the CPU, while the the dedicated GPU requires transferring the data from the host to the GPU device. However, the dedicated GPU shows better scalability for more characters, thanks to the large number of cores available. Note that the rendering time is not included in these numbers.

In summary, our algorithm is suitable to parallelize and can animate more than 500 characters in real-time on deformable terrains. Results can be seen in Fig. 8 .

8 Discussions and Conclusion

The carpet unrolling approach is highly adaptive and can apply a single type of locomotion to various terrains with different geometry. Compared to classic methods based on inverse kinematics, there is no need to specify when the feet are in contact with the ground. The body movements are automatically computed from the geometry of the terrain, and therefore there are very few parameters to tune to produce natural movements. The method is highly parallelizable and can be implemented on the GPU, resulting in hundreds of characters animated in real-time. In the future, we plan to enhance the method to simulate character-character interaction as well as character-obstacle interactions.

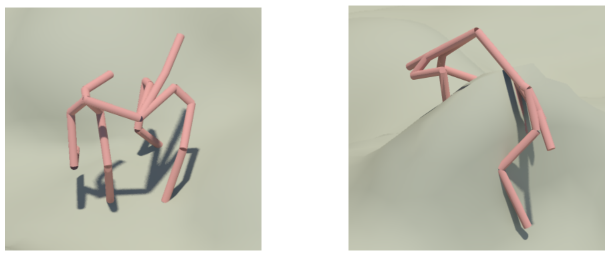


Figure 9: *The method can suffer from very sharp turns (left) and very steep terrains (right).*

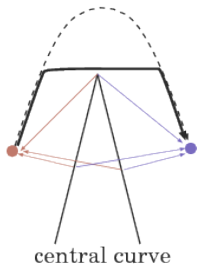


Figure 10: *A sharp turn of the central line will result in a fast translation of the joints when the body passes the acute region of the central line.*

Limitations Although the characters can adapt well to the terrain, the method may suffer from sharp turns or very steep terrains, as can be observed in Fig. 9. When such extreme cases happen, the linear interpolation of the relative vectors result in bad movements. For example, assume the central curve is turned very sharply as shown in Fig. 10. As the offsets of the descriptor points are made to be perpendicular to the central curve, the offset vector suddenly change its direction when passing the acute region. As a result, the trajectory of the joints farther from the central line will quickly translate when the body passes the acute region, as shown in the bold line of Fig. 10, where the ideal trajectory is a mild curve drawn by the dashed line. A better scheme to locate the descriptor points is needed to handle such a case.

Also, our system does not have any collision avoidance framework between characters. This will involve re-planning the carpet trajec-

Setup	1	10	20	30	64	128	256	512
CPU single thread	7.84 (0.67)	14.33 (6.57)	18.36 (10.43)	24.04 (15.79)	42.1 (33.27)	83.12 (73.4)	166.95 (155.87)	279.17 (266.3)
CPU multi-thread	9.29 (0.29)	9.71 (0.89)	11 (1.5)	11.8 (1.96)	13.95 (3.99)	18.14 (7.89)	27.35 (15.8)	43.93 (32)
GPU integrated	10.17 (0.54)	10.28 (0.9)	10.24 (1.22)	10.8 (1.83)	13.97 (4.09)	20.51 (8.7)	29.31 (16.48)	42.18 (29.5)
GPU dedicated	10 (1.75)	11.62 (2.33)	12.44 (2.87)	13.2 (3.44)	13.68 (3.57)	15.03 (4.14)	19.17 (6.68)	23.64 (12.26)

Table 1: *The computation time* (ms) *for one frame in different setups. The numbers in the brackets are those spent by the kernel, excluding the transmission time of the data.*

tory on the fly.

References

AL-ASQHAR, R. A., KOMURA, T., AND CHOI, M. G. 2013. Relationship descriptors for interactive motion adaptation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, 45–53.

GLEICHER, M. 1997. Motion editing with spacetime constraints. *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics*.

HELBING, D., FARKAS, I., AND VICSEK, T. 2000. Simulating dynamical features of escape panic. *Nature 407*, 487–490.

HO, E. S. L., KOMURA, T., AND TAI, C.-L. 2010. Spatial relationship preserving character motion adaptation. *ACM Transactions on Graphics 29*, 4.

JAKOBSEN, T. 2001. Advanced character physics. *In Game Developers Conference Proceedings*, 383–401.

KOVAR, L., AND GLEICHER, M. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics 23*, 3, 559–568.

LEE, J., AND SHIN, S. Y. 1999. A hierarchical approach to interactive motion editing for human-like figures. *Proceedings of SIGGRAPH '99*, 39–48.

MUKAI, T., AND KURIYAMA, S. 2005. Geostatistical motion interpolation. *ACM Trans. Graph. 24*, 3, 1062–1070.

MUKAI, T. 2011. Motion rings for interactive gait synthesis. In *Symposium on Interactive 3D Graphics and Games*, ACM, 125–132.

ONDREJ, J., PETTR, J., OLIVIER, A.-H., AND DONIKIAN, S. 2010. A synthetic-vision-based steering approach for crowd simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010) 29*, 4.

REYNOLDS, C. 1987. Flocks, herds, and schools: A distributed behavioral model. *Proceedings of SIGGRAPH 87 21*, 25–34.

ROSE, C., COHEN, M. F., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl. 18*, 5, 32–40.

SHIN, H. J., LEE, J., SHIN, S. Y., AND GLEICHER, M. 2001. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics 20*, 2, 67–94.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. *ACM Transactions on Graphics (TOG) 25*, 3, 1160–1168.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of multiple agents in crowded environments. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 139–147.